Assignment 6

Group 5

Student Name: Zhihao Ren, Chu Gong, Juhi Singhal, Zefu Chen

1.

In [1]:

```
from libsettings import *
import yfinance as yf
from yahoofinancials import YahooFinancials
set_stuff()
```

S&P 500 Part

In [2]:

```
spx_df = yf.download('^GSPC', start='1970-01-01', end='2021-05-01', progress=False)
spx_df.head(10)
```

Out[2]:

Date	Open	High	Low	Close	Adj Close	Volume
1970-01-02	0.000	93.540	91.790	93.000	93.000	8050000
1970-01-05	0.000	94.250	92.530	93.460	93.460	11490000
1970-01-06	0.000	93.810	92.130	92.820	92.820	11460000
1970-01-07	0.000	93.380	91.930	92.630	92.630	10010000
1970-01-08	0.000	93.470	91.990	92.680	92.680	10670000
1970-01-09	0.000	93.250	91.820	92.400	92.400	9380000
1970-01-12	0.000	92.670	91.200	91.700	91.700	8900000
1970-01-13	0.000	92.610	90.990	91.920	91.920	9870000
1970-01-14	0.000	92.400	90.880	91.650	91.650	10380000
1970-01-15	0.000	92.350	90.730	91.680	91.680	11120000

In [3]:

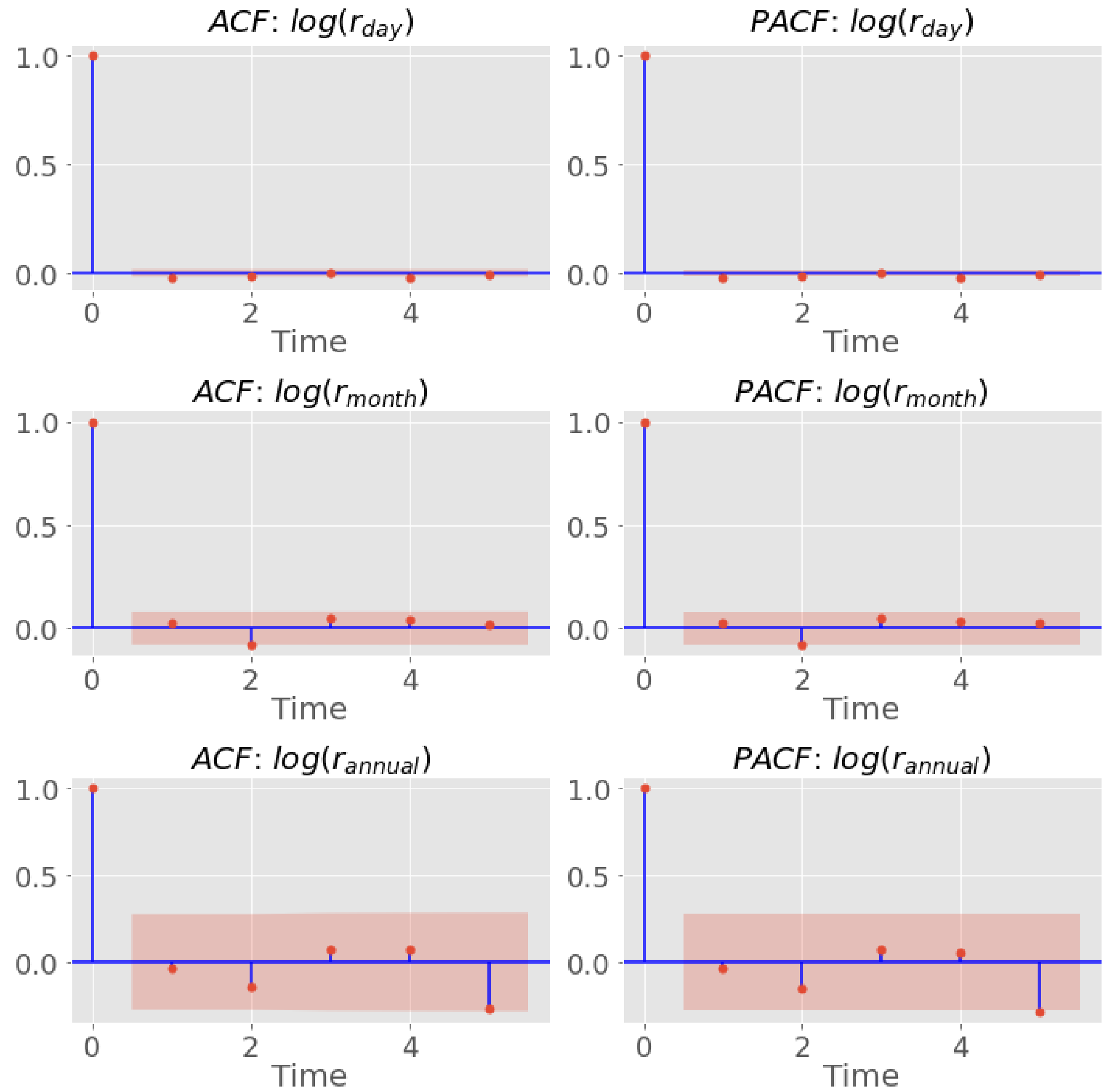
```
spx_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 12948 entries, 1970-01-02 to 2021-04-30
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Open       12948 non-null  float64
 1   High       12948 non-null  float64
 2   Low        12948 non-null  float64
 3   Close      12948 non-null  float64
 4   Adj Close  12948 non-null  float64
 5   Volume     12948 non-null  int64  
dtypes: float64(5), int64(1)
memory usage: 708.1 KB
```

(a)

In [4]:

```
# compute log-return with different freq
ret_daily = np.log(spx_df['Close']).diff().dropna()
ret_monthly = np.log(spx_df.resample('1m').first()['Close']).diff().dropna()
ret_annual = np.log(spx_df.resample('1y').first()['Close']).diff().dropna()

# plot autocorrelation
nlags = 5
fig, axes = plt.subplots(3, 2, figsize=(10, 10))
plot_acf(ret_daily, ax=axes[0][0], lags=nlags)
plot_pacf(ret_daily, ax=axes[0][1], lags=nlags)
plot_acf(ret_monthly, ax=axes[1][0], lags=nlags)
plot_pacf(ret_monthly, ax=axes[1][1], lags=nlags)
plot_acf(ret_annual, ax=axes[2][0], lags=nlags)
plot_pacf(ret_annual, ax=axes[2][1], lags=nlags)

axes[0][0].set_xlabel('Time', fontsize=20)
axes[0][0].set_title("${\it ACF}$: $log(r_{day})$", fontsize=20)
axes[0][1].set_xlabel('Time', fontsize=20)
axes[0][1].set_title("${\it PACF}$: $log(r_{day})$", fontsize=20)
axes[1][0].set_xlabel('Time', fontsize=20)
axes[1][0].set_title("${\it ACF}$: $log(r_{month})$", fontsize=20)
axes[1][1].set_xlabel('Time', fontsize=20)
axes[1][1].set_title("${\it PACF}$: $log(r_{month})$", fontsize=20)
axes[2][0].set_xlabel('Time', fontsize=20)
axes[2][0].set_title("${\it ACF}$: $log(r_{annual})$", fontsize=20)
axes[2][1].set_xlabel('Time', fontsize=20)
axes[2][1].set_title("${\it PACF}$: $log(r_{annual})$", fontsize=20)
fig.tight_layout()
```



(b)

In [5]:

```
# compute average volatility
vol_ann_day = np.std(ret_daily) * np.sqrt(252)
vol_ann_mon = np.std(ret_monthly) * np.sqrt(12)
vol_ann_ann = np.std(ret_annual)

print(f"Average annualized volatility of daily return: {vol_ann_day}")
print(f"Average annualized volatility of monthly return: {vol_ann_mon}")
print(f"Average annualized volatility of annual return: {vol_ann_ann}")
```

```
Average annualized volatility of daily return: 0.1725434114608058
Average annualized volatility of monthly return: 0.15835887231104673
Average annualized volatility of annual return: 0.1586393190851782
```

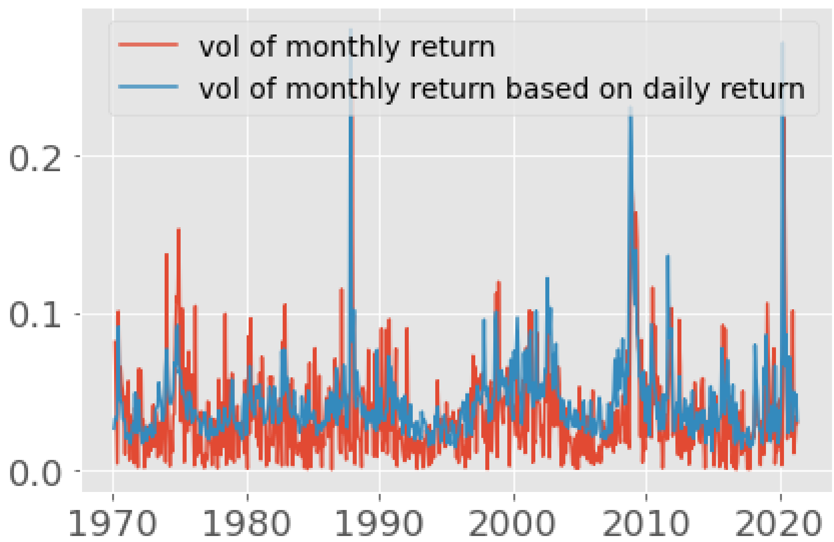
From above results, it can be showed that the average annualized volatility of daily return is different from the ones computed using monthly return and annual return, which results from the fact that the annualized volatility is an approxiamation based on the assumption that the data is i.i.d., while i.i.d. assumption cannot be hold in this case.

(c)

In [6]:

```
vol_mon_mon = np.abs(ret_monthly)
vol_mon_day = ret_daily.resample('1m').std() * np.sqrt(21)

fig, ax = plt.subplots()
ax.plot(vol_mon_mon, label='vol of monthly return')
ax.plot(vol_mon_day, label='vol of monthly return based on daily return')
plt.legend()
fig.tight_layout()

print("Mean of vol of monthly return based on daily return: {}".format(vol_mon_day.mean()))
print("Variance of vol of monthly return based on daily return: {}".format(vol_mon_day.var(
print("Mean of vol of monthly return: {}".format(vol_mon_mon.mean()))
print("Variance of vol of monthly return: {}".format(vol_mon_mon.var()))
print("Correlation of two series: {}".format(vol_mon_day.corr(vol_mon_mon)))
```

```
Mean of vol of monthly return based on daily return: 0.042535938839692984
Variance of vol of monthly return based on daily return: 0.00068460840429693
71
Mean of vol of monthly return: 0.034432962337145184
Variance of vol of monthly return: 0.0009432054133356655
Correlation of two series: 0.41791439108721495
```



(d)

In [7]:

```
ret_daily_sq = np.power(ret_daily, 2)
realized_vol = np.sqrt(ret_daily_sq.resample('1m').sum())

y = realized_vol.values[1:]
x = realized_vol.values[:-1]
reg = sm.OLS(y, sm.add_constant(x)).fit(cov_type='HC0')
print(reg.summary())
```

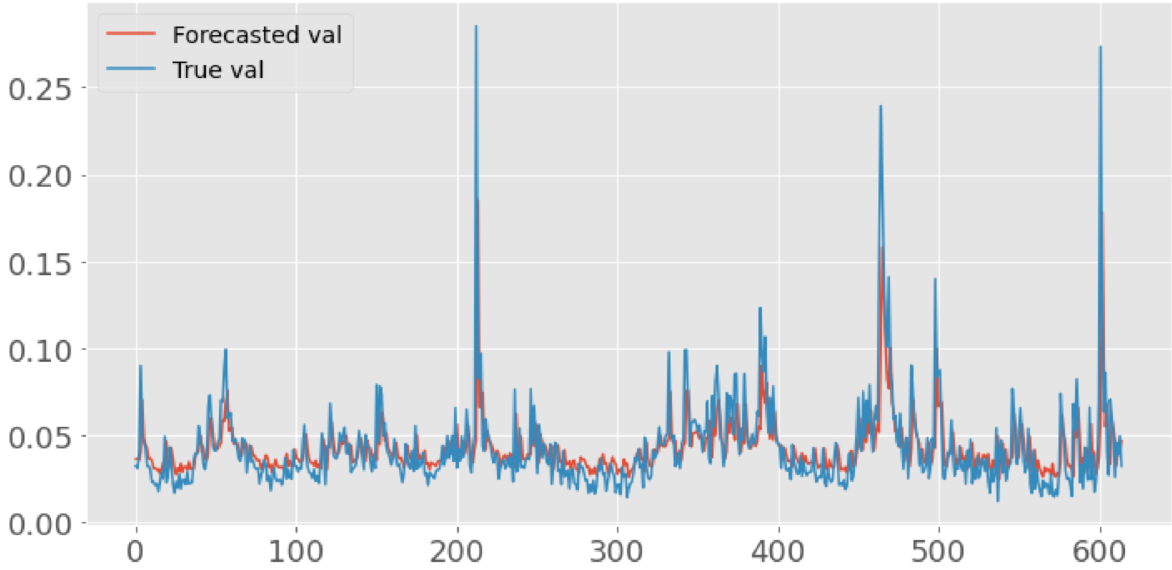
```
                            OLS Regression Results
============================================================================
==
Dep. Variable:                      y   R-squared:                       0.3
46
Model:                            OLS   Adj. R-squared:                  0.3
44
Method:                 Least Squares   F-statistic:                     50.
77
Date:                Mon, 03 May 2021   Prob (F-statistic):           2.93e-
12
Time:                        16:44:41   Log-Likelihood:                 150
1.7
No. Observations:                 615   AIC:                            -299
9.
Df Residuals:                     613   BIC:                            -299
1.
Df Model:                           1
Covariance Type:                  HC0
============================================================================
==
                 coef    std err          z      P>|z|      [0.025      0.97
5]
----------------------------------------------------------------------------
--
const          0.0175      0.003      5.461      0.000       0.011       0.0
24
x1             0.5878      0.082      7.125      0.000       0.426       0.7
49
============================================================================
==
Omnibus:                      670.975   Durbin-Watson:                   2.1
67
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            60055.4
48
Skew:                           4.925   Prob(JB):                         0.
00
Kurtosis:                      50.398   Cond. No.                         3
8.5
============================================================================
==

Notes:
[1] Standard Errors are heteroscedasticity robust (HC0)
```

From the regression, we can see that the coef is significantly positive (i.e. ~0.58), and the R^2 is 0.346. One thing should be noticed is that the kurtosis is 50.398, which suggests a fat tail there.

(e)

In [8]:

```
fig, ax = plt.subplots(figsize=(10, 5))
ax.plot(reg.predict(), label='Forecasted val')
ax.plot(y, label='True val')
plt.legend()
fig.tight_layout()
```



(f)

In [9]:

```
res = np.mean(np.power(reg.resid, 2))
print(f"Mean Squared Error: {res}")
```

```
Mean Squared Error: 0.00044315856541676763
```

From the forecast result, it can be seen that the prediction is not good, the norm of the spike cannot be perfectly catched, also the realized vol is more oscillated than the predicted one.

Now we go the same route for GE.

GE Part

In [10]:

```
ge_df = yf.download('GE', start='1970-01-01', end='2021-05-01', progress=False)
ge_df.head(10)
```

Out[10]:

	Open	High	Low	Close	Adj Close	Volume
Date						
1970-01-02	0.776	0.777	0.766	0.767	0.159	2316288
1970-01-05	0.767	0.771	0.757	0.764	0.159	4233216
1970-01-06	0.762	0.762	0.737	0.741	0.154	3544320
1970-01-07	0.744	0.755	0.744	0.745	0.155	4602624
1970-01-08	0.747	0.759	0.747	0.751	0.156	13897728
1970-01-09	0.751	0.752	0.732	0.732	0.152	5940480
1970-01-12	0.732	0.735	0.725	0.731	0.152	4043520
1970-01-13	0.731	0.737	0.730	0.734	0.152	2875392
1970-01-14	0.735	0.752	0.735	0.747	0.155	3694080
1970-01-15	0.747	0.754	0.746	0.749	0.155	2675712

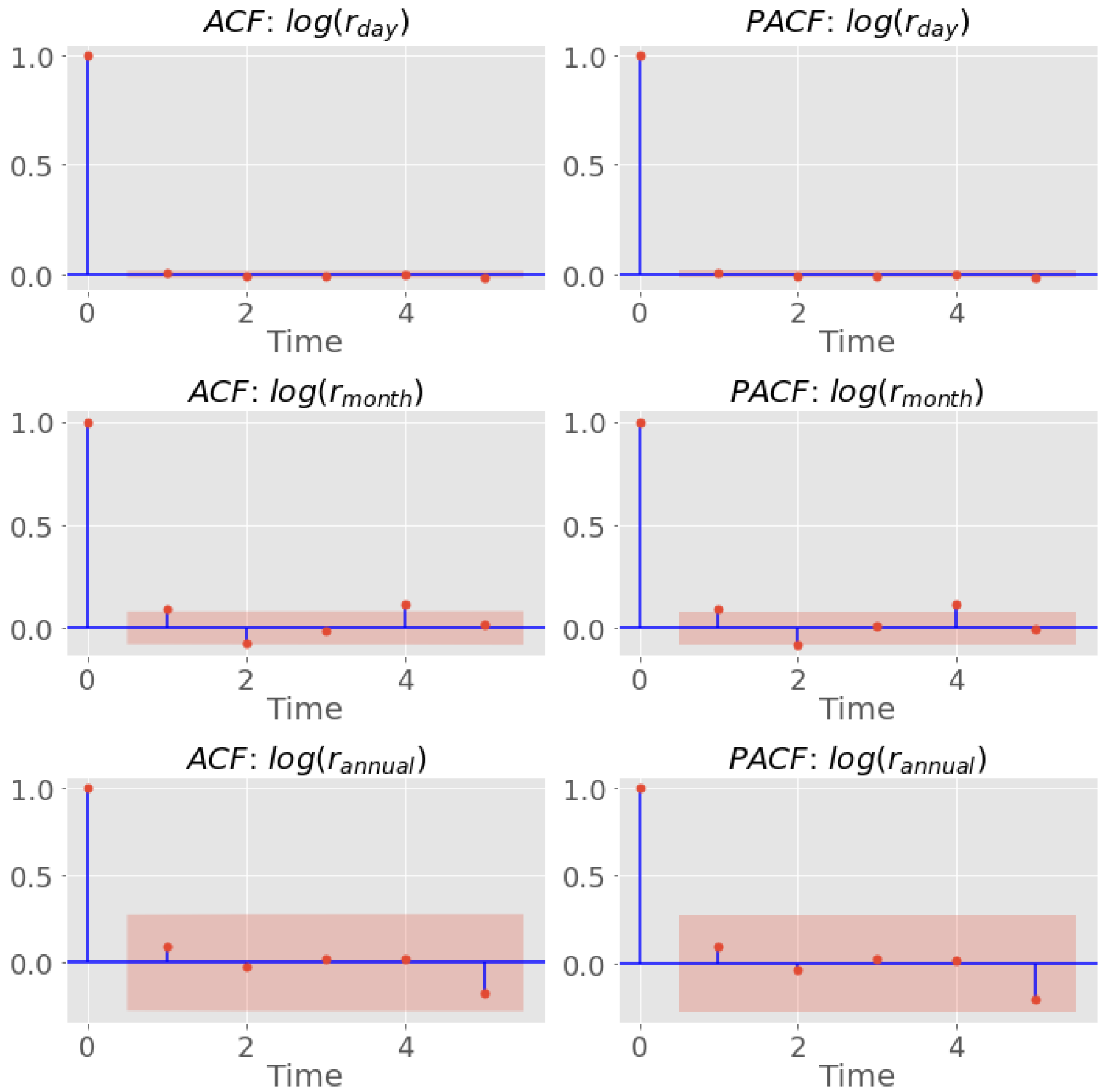
In [11]:

```
ge_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 12948 entries, 1970-01-02 to 2021-04-30
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Open       12948 non-null  float64
 1   High       12948 non-null  float64
 2   Low        12948 non-null  float64
 3   Close      12948 non-null  float64
 4   Adj Close  12948 non-null  float64
 5   Volume     12948 non-null  int64  
dtypes: float64(5), int64(1)
memory usage: 708.1 KB
```

In [12]:

```
# compute log-return with different freq
ret_daily = np.log(ge_df['Close']).diff().dropna()
ret_monthly = np.log(ge_df.resample('1m').first()['Close']).diff().dropna()
ret_annual = np.log(ge_df.resample('1y').first()['Close']).diff().dropna()

# plot autocorrelation
nlags = 5
fig, axes = plt.subplots(3, 2, figsize=(10, 10))
plot_acf(ret_daily, ax=axes[0][0], lags=nlags)
plot_pacf(ret_daily, ax=axes[0][1], lags=nlags)
plot_acf(ret_monthly, ax=axes[1][0], lags=nlags)
plot_pacf(ret_monthly, ax=axes[1][1], lags=nlags)
plot_acf(ret_annual, ax=axes[2][0], lags=nlags)
plot_pacf(ret_annual, ax=axes[2][1], lags=nlags)

axes[0][0].set_xlabel('Time', fontsize=20)
axes[0][0].set_title("${\it ACF}$: $log(r_{day})$", fontsize=20)
axes[0][1].set_xlabel('Time', fontsize=20)
axes[0][1].set_title("${\it PACF}$: $log(r_{day})$", fontsize=20)
axes[1][0].set_xlabel('Time', fontsize=20)
axes[1][0].set_title("${\it ACF}$: $log(r_{month})$", fontsize=20)
axes[1][1].set_xlabel('Time', fontsize=20)
axes[1][1].set_title("${\it PACF}$: $log(r_{month})$", fontsize=20)
axes[2][0].set_xlabel('Time', fontsize=20)
axes[2][0].set_title("${\it ACF}$: $log(r_{annual})$", fontsize=20)
axes[2][1].set_xlabel('Time', fontsize=20)
axes[2][1].set_title("${\it PACF}$: $log(r_{annual})$", fontsize=20)
fig.tight_layout()
```



In [13]:

```
# compute average volatility
vol_ann_day = np.std(ret_daily) * np.sqrt(252)
vol_ann_mon = np.std(ret_monthly) * np.sqrt(12)
vol_ann_ann = np.std(ret_annual)

print(f"Average annualized volatility of daily return: {vol_ann_day}")
print(f"Average annualized volatility of monthly return: {vol_ann_mon}")
print(f"Average annualized volatility of annual return: {vol_ann_ann}")
```

```
Average annualized volatility of daily return: 0.2831381612138961
Average annualized volatility of monthly return: 0.2691433654414752
Average annualized volatility of annual return: 0.2886513753205717
```

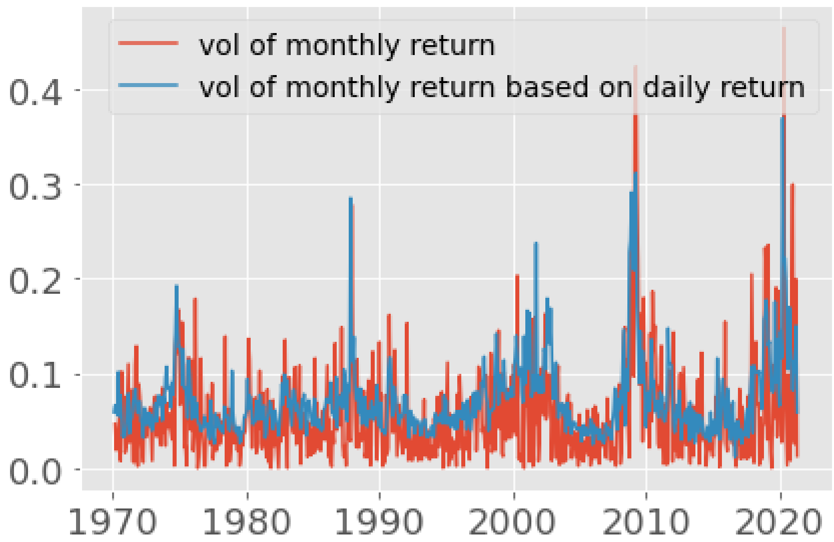
From above results, it can be showed that the average annualized volatility are slightly different , which results from the fact that the annualized volatility is an approxiamation based on the assumption that the data is i.i.d., while i.i.d. assumption cannot be hold in this case.

(c)

In [14]:

```
vol_mon_mon = np.abs(ret_monthly)
vol_mon_day = ret_daily.resample('1m').std() * np.sqrt(21)

fig, ax = plt.subplots()
ax.plot(vol_mon_mon, label='vol of monthly return')
ax.plot(vol_mon_day, label='vol of monthly return based on daily return')
plt.legend()
fig.tight_layout()

print("Mean of vol of monthly return based on daily return: {}".format(vol_mon_day.mean()))
print("Variance of vol of monthly return based on daily return: {}".format(vol_mon_day.var(
print("Mean of vol of monthly return: {}".format(vol_mon_mon.mean()))
print("Variance of vol of monthly return: {}".format(vol_mon_mon.var()))
print("Correlation of two series: {}".format(vol_mon_day.corr(vol_mon_mon)))
```

```
Mean of vol of monthly return based on daily return: 0.07143836967255703
Variance of vol of monthly return based on daily return: 0.0016347282800572812
Mean of vol of monthly return: 0.056031956074503536
Variance of vol of monthly return: 0.0029231745718070103
Correlation of two series: 0.45446269609319506
```



In [15]:

```
ret_daily_sq = np.power(ret_daily, 2)
realized_vol = np.sqrt(ret_daily_sq.resample('1m').sum())

y = realized_vol.values[1:]
x = realized_vol.values[:-1]
reg = sm.OLS(y, sm.add_constant(x)).fit(cov_type='HC0')
print(reg.summary())
```

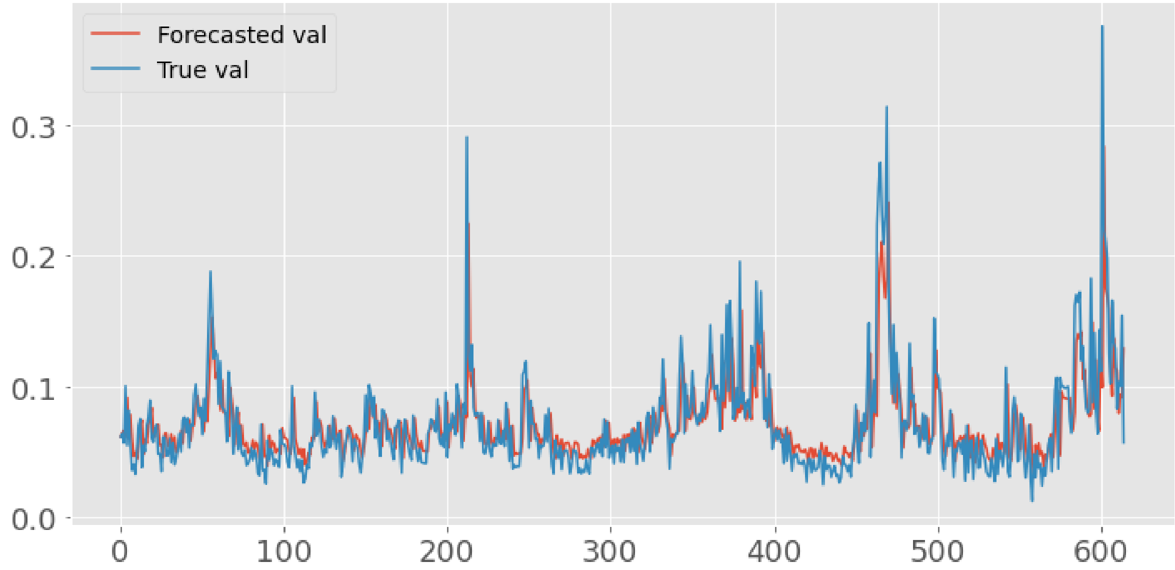
```
                            OLS Regression Results
============================================================================
==
Dep. Variable:                      y   R-squared:                       0.4
87
Model:                            OLS   Adj. R-squared:                  0.4
87
Method:                 Least Squares   F-statistic:                      18
1.6
Date:                Mon, 03 May 2021   Prob (F-statistic):           1.91e-
36
Time:                        16:44:43   Log-Likelihood:                  130
9.9
No. Observations:                 615   AIC:                            -261
6.
Df Residuals:                     613   BIC:                            -260
7.
Df Model:                           1
Covariance Type:                  HC0
============================================================================
==
                 coef    std err          z      P>|z|      [0.025      0.97
5]
----------------------------------------------------------------------------
--
const          0.0215      0.003      6.592      0.000       0.015       0.0
28
x1             0.6981      0.052     13.477      0.000       0.597       0.8
00
============================================================================
==
Omnibus:                      449.691   Durbin-Watson:                   2.3
94
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            11839.3
40
Skew:                           2.906   Prob(JB):                         0.
00
Kurtosis:                      23.694   Cond. No.                         2
5.0
============================================================================
==

Notes:
[1] Standard Errors are heteroscedasticity robust (HC0)
```

From the regression, we can see that the coef is significantly positive (i.e. ~0.698), and the R^2 is 0.487. One thing should be noticed is that the kurtosis is 23.694, which suggests a fat tail there.

(e)

In [16]:

```
fig, ax = plt.subplots(figsize=(10, 5))
ax.plot(reg.predict(), label='Forecasted val')
ax.plot(y, label='True val')
plt.legend()
fig.tight_layout()
```



(f)

In [17]:

```
res = np.mean(np.power(reg.resid, 2))
print(f"Mean Squared Error: {res}")
```

```
Mean Squared Error: 0.0008271252767787558
```

From the forecast result, it can be seen that the prediction is not good, the norm of the spike cannot be perfectly catched, also the realized vol is more oscillated than the predicted one.

In [ ]:

PS6-out-of-sample-test

May 5, 2021

0.0.1 Q2

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        sns.set()
        import yfinance as yf
        import arch
        from arch.univariate import ConstantMean, GARCH, Normal
        from arch import arch_model
        from statsmodels.tsa.arima.model import ARIMA
        import statsmodels.api as sm
        import warnings
        warnings.filterwarnings("ignore")
```

```
In [2]: df_sp500_ret = yf.download("^GSPC", start="1970-01-01", end="2021-05-01",
                                   progress=False)[["Adj Close"]]
        df_ge_ret = yf.download("GE", start="1970-01-01", end="2021-05-01",
                                progress=False)[["Adj Close"]]
```

We will present all question results for S&P500 first, and then GE.

```
In [3]: # monthly log return
        df_sp500_ret["yearmonth"] = df_sp500_ret.index.astype(str).str[:7]
        df_sp500_ret_monthly = df_sp500_ret.drop_duplicates(subset=["yearmonth"], keep="last")
        df_sp500_ret_monthly["log_Close"] = np.log(df_sp500_ret_monthly["Adj Close"])
        df_sp500_ret_monthly["log_return"] = 100*(df_sp500_ret_monthly["log_Close"] - \
                                                  df_sp500_ret_monthly["log_Close"].shift(1))
```

```
In [4]: arr_monthly_ret = df_sp500_ret_monthly["log_return"].dropna()
```

(a) GARCH(1, 1)

```
In [5]: # out-of-sample forecast
        split_date = "2014-12-31"

        am = ConstantMean(arr_monthly_ret[:split_date])
```

```
# am = ConstantMean(arr_monthly_ret)
am.volatility = GARCH(1, 0, 1)
am.distribution = Normal()
res = am.fit(disp="off")

print(res.summary())
```

```
                     Constant Mean - GARCH Model Results
==============================================================================
Dep. Variable:             log_return   R-squared:                       0.000
Mean Model:             Constant Mean   Adj. R-squared:                  0.000
Vol Model:                      GARCH   Log-Likelihood:               -1551.91
Distribution:                  Normal   AIC:                           3111.82
Method:            Maximum Likelihood   BIC:                           3128.99
                                        No. Observations:                  540
Date:                Wed, May 05 2021   Df Residuals:                      539
Time:                        21:54:14   Df Model:                            1
                               Mean Model
========================================================================
                 coef    std err          t      P>|t|  95.0% Conf. Int.
------------------------------------------------------------------------
mu             0.6483      0.185      3.506  4.541e-04 [  0.286,  1.011]
                              Volatility Model
==========================================================================
                 coef    std err          t      P>|t|    95.0% Conf. Int.
--------------------------------------------------------------------------
omega          0.8052      0.513      1.568      0.117   [ -0.201,  1.812]
alpha[1]       0.1227  3.354e-02      3.657  2.547e-04 [5.693e-02,  0.188]
beta[1]        0.8442  3.468e-02     24.342 7.067e-131   [  0.776,  0.912]
==========================================================================

Covariance estimator: robust
```

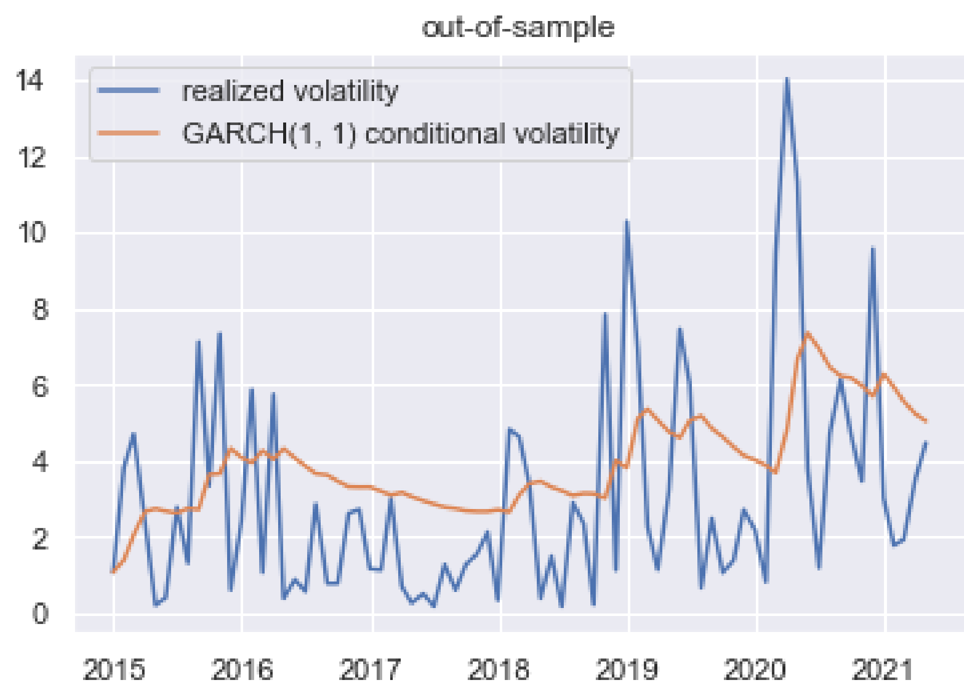
From the GARCH(1, 1) estimated results, we noted that alpha + beta is close to 1, which means that the volatility of monthly return is highly persistent.

(b) realized monthly volatility and conditional GARCH(1,1) volatility

```
In [6]: arr_ret_error = arr_monthly_ret[split_date:] - res.params["mu"]
        arr_realized_vol = np.abs(arr_ret_error)
        arr_conditional_vol = pd.Series(index=arr_realized_vol.index)
        arr_conditional_vol[split_date] = arr_realized_vol[split_date]

        for i in range(1, len(arr_conditional_vol[split_date:])):
            arr_conditional_vol[i] = np.sqrt(res.params["omega"]
                    + res.params["alpha[1]"]*arr_realized_vol[i-1]**2
                    + res.params["beta[1]"]*arr_conditional_vol[i-1]**2)
```

```
# arr_conditional_vol = res.conditional_volatility
plt.title("out-of-sample")
plt.plot(arr_realized_vol, label="realized volatility")
plt.plot(arr_conditional_vol, label="GARCH(1, 1) conditional volatility")
plt.legend()
```

Out[6]: <matplotlib.legend.Legend at 0x12a7a7cf8>

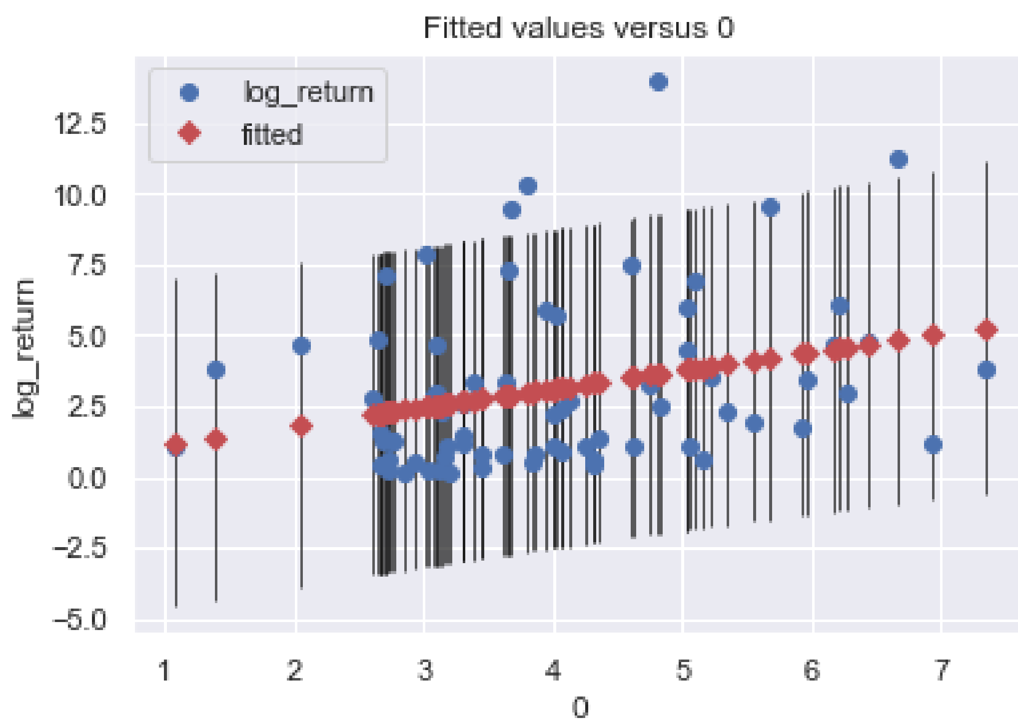


In [7]:
```
model = sm.OLS(arr_realized_vol, sm.add_constant(arr_conditional_vol))
results = model.fit()
results.summary()
```

Out[7]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:             log_return   R-squared:                       0.083
Model:                            OLS   Adj. R-squared:                  0.071
Method:                 Least Squares   F-statistic:                     6.785
Date:                Wed, 05 May 2021   Prob (F-statistic):             0.0111
Time:                        21:54:20   Log-Likelihood:                -187.83
No. Observations:                  77   AIC:                             379.7
Df Residuals:                      75   BIC:                             384.3
Df Model:                           1
Covariance Type:            nonrobust
```

```
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.5159      1.031      0.500      0.618      -1.538       2.570
0              0.6469      0.248      2.605      0.011       0.152       1.142
==============================================================================
Omnibus:                       26.142   Durbin-Watson:                   1.552
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               38.463
Skew:                           1.447   Prob(JB):                     4.45e-09
Kurtosis:                       4.900   Cond. No.                         14.1
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly speci
"""
```

```
In [8]: sm.graphics.plot_fit(results,1)
        plt.show()
```



(c) RMSE of GARCH forecasts

```
In [9]: RMSE = np.sqrt(((arr_realized_vol - arr_conditional_vol)**2).sum()/ \
                    len(arr_conditional_vol))

        RMSE
```

Out[9]: 2.9450054362039064

(d) find the best GARCH(p, q) model

```
In [10]: df_model_OS_results = pd.DataFrame(
             index=["BIC", "RMSE", "MAE", "MAPE"],
             columns=["GARCH({}, {})".format(p+1, q+1) for p in range(3)
                      for q in range(3)])
```

```
In [11]: for p in range(1, 4):
             for q in range(1, 4):
                 split_date = "2014-12-31"
                 am = ConstantMean(arr_monthly_ret[:split_date])
                 # am = ConstantMean(arr_monthly_ret)
                 am.volatility = GARCH(p, 0, q)
                 am.distribution = Normal()
                 res = am.fit(disp="off")

                 garch_name = "GARCH({}, {})".format(p, q)
                 df_model_OS_results.loc["BIC", garch_name] = res.bic

                 # print(res.summary())
                 arr_ret_error = arr_monthly_ret[split_date:] - res.params["mu"]
                 arr_realized_vol = np.abs(arr_ret_error)
                 arr_conditional_vol = pd.Series(index=arr_realized_vol.index)
                 # arr_conditional_vol[split_date] = arr_realized_vol[split_date]
                 arr_conditional_vol[:max(p, q)] = arr_realized_vol[:max(p, q)]

                 for i in range(max(p, q), len(arr_conditional_vol[split_date:])):
                     step_sum = res.params["omega"]
                     for j in range(p):
                         step_sum += \
                             res.params["alpha[{}]".format(j+1)]*arr_realized_vol[i-1-j]**2
                     for k in range(q):
                         step_sum += \
                             res.params["beta[{}]".format(k+1)]*arr_conditional_vol[i-1-k]**2
                     arr_conditional_vol[i] = np.sqrt(step_sum)

                 df_model_OS_results.loc["RMSE", garch_name] = \
                     np.sqrt(((arr_realized_vol - arr_conditional_vol)**2).sum()/ \
                         len(arr_conditional_vol))
                 df_model_OS_results.loc["MAE", garch_name] = \
                     np.abs(arr_realized_vol - arr_conditional_vol).sum()/ \
                         len(arr_conditional_vol)
                 df_model_OS_results.loc["MAPE", garch_name] = np.abs(
                     100*(arr_realized_vol - arr_conditional_vol)/arr_realized_vol).sum()/ \
                         len(arr_conditional_vol)
```

```
In [12]: df_model_OS_results
```

```
Out[12]:      GARCH(1, 1) GARCH(1, 2) GARCH(1, 3) GARCH(2, 1) GARCH(2, 2) GARCH(2, 3)  \
         BIC      3128.99      3134.4      3140.7     3133.71     3139.24     3145.53
         RMSE     2.94501     2.94412     2.94998     3.03346     2.94079     2.94521
         MAE      2.46352     2.46693     2.47519     2.51738     2.46135     2.46463
         MAPE      259.01      291.81     322.253     284.312     265.055     285.198

              GARCH(3, 1) GARCH(3, 2) GARCH(3, 3)
         BIC      3138.17     3143.62     3139.75
         RMSE     3.08856      3.0858      3.1196
         MAE      2.57498     2.55924     2.52404
         MAPE     279.237     276.045     271.708
```

From the BIC perspective, GARCH(1,1) has the smallest BIC. But GARCH(1, 2) has lower forcast error. To keep model parsimonious, we choose GARCH(1,1).

0.0.2 Q3 GJR-GARCH model

(a) GJR-GARCH(1, 1)

```
In [13]: am = arch_model(arr_monthly_ret[:split_date], p=1, o=1, q=1)
         res = am.fit(disp="off")
         print(res.summary())
```

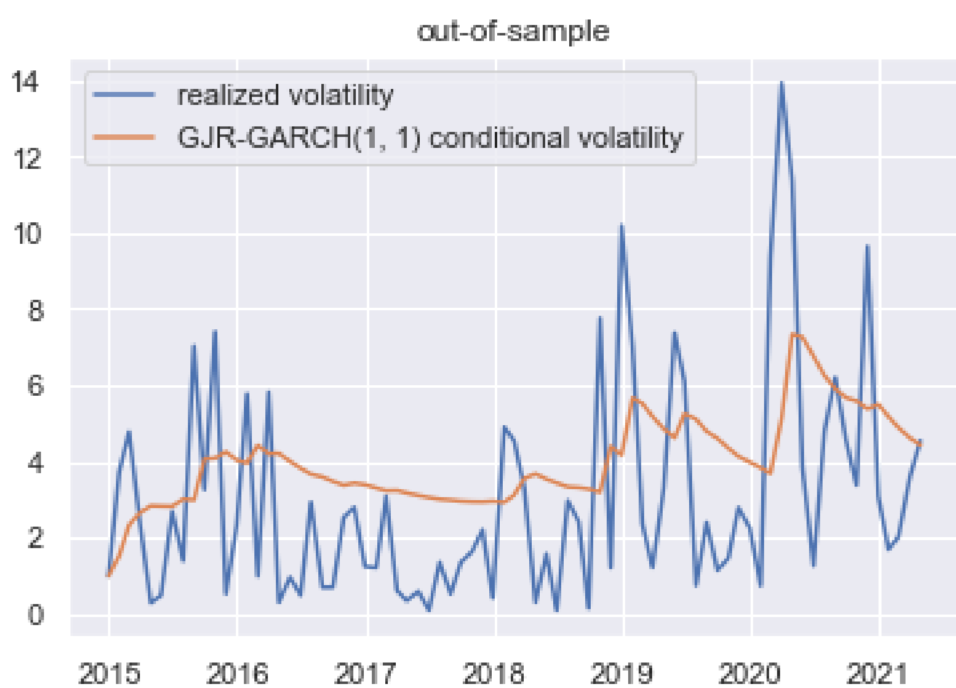
```
                   Constant Mean - GJR-GARCH Model Results
==============================================================================
Dep. Variable:             log_return   R-squared:                       0.000
Mean Model:             Constant Mean   Adj. R-squared:                  0.000
Vol Model:                  GJR-GARCH   Log-Likelihood:               -1549.93
Distribution:                  Normal   AIC:                           3109.85
Method:            Maximum Likelihood   BIC:                           3131.31
                                        No. Observations:                  540
Date:                Wed, May 05 2021   Df Residuals:                      539
Time:                        21:54:43   Df Model:                            1
                               Mean Model
========================================================================
                 coef    std err          t      P>|t|  95.0% Conf. Int.
------------------------------------------------------------------------
mu             0.5673      0.175      3.244  1.178e-03 [  0.225,  0.910]
                            Volatility Model
========================================================================
                 coef    std err          t      P>|t|  95.0% Conf. Int.
------------------------------------------------------------------------
omega          1.3454      2.956      0.455      0.649 [ -4.449,  7.140]
alpha[1]       0.0536      0.152      0.353      0.724 [ -0.244,  0.351]
gamma[1]       0.1045      0.209      0.501      0.616 [ -0.304,  0.513]
beta[1]        0.8254      0.135      6.119  9.410e-10 [  0.561,  1.090]
========================================================================
```

```
Covariance estimator: robust
```

(b) realized monthly volatility and conditional GJR-GARCH(1,1) volatility

```
In [14]: arr_ret_error = arr_monthly_ret[split_date:] - res.params["mu"]
         arr_realized_vol = np.abs(arr_ret_error)
         arr_conditional_vol = pd.Series(index=arr_realized_vol.index)
         arr_conditional_vol[split_date] = arr_realized_vol[split_date]

         for i in range(1, len(arr_conditional_vol[split_date:])):
             temp_sum = res.params["omega"] \
                     + res.params["alpha[1]"]*arr_realized_vol[i-1]**2 \
                     + res.params["beta[1]"]*arr_conditional_vol[i-1]**2
             if arr_ret_error[i-1]<0:
                 temp_sum += res.params["gamma[1]"]*arr_realized_vol[i-1]**2

             arr_conditional_vol[i] = np.sqrt(temp_sum)

         # arr_conditional_vol = res.conditional_volatility
         plt.title("out-of-sample")
         plt.plot(arr_realized_vol, label="realized volatility")
         plt.plot(arr_conditional_vol, label="GJR-GARCH(1, 1) conditional volatility")
         plt.legend()
```

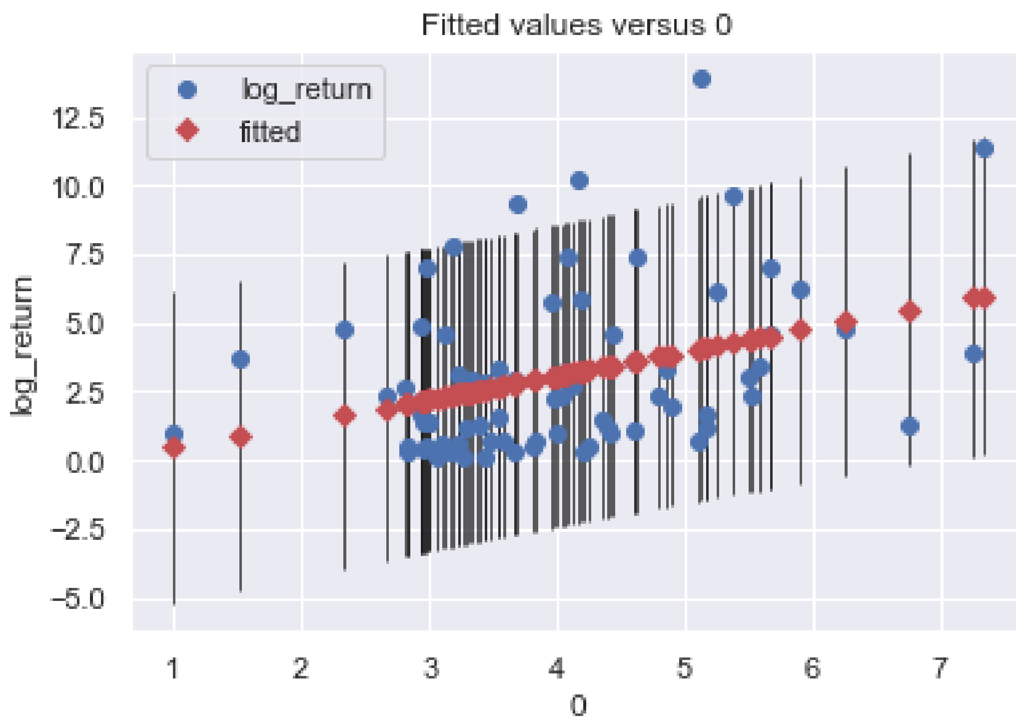
```
Out[14]: <matplotlib.legend.Legend at 0x11b5c27f0>
```



In [15]:
```
model = sm.OLS(arr_realized_vol, sm.add_constant(arr_conditional_vol))
results = model.fit()
results.summary()
```

Out[15]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:             log_return   R-squared:                       0.125
Model:                            OLS   Adj. R-squared:                  0.113
Method:                 Least Squares   F-statistic:                     10.72
Date:                Wed, 05 May 2021   Prob (F-statistic):            0.00160
Time:                        21:54:45   Log-Likelihood:                -185.92
No. Observations:                  77   AIC:                             375.8
Df Residuals:                      75   BIC:                             380.5
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.3960      1.107     -0.358      0.721      -2.601       1.809
0              0.8690      0.265      3.274      0.002       0.340       1.398
==============================================================================
Omnibus:                       22.527   Durbin-Watson:                   1.632
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               30.217
Skew:                           1.321   Prob(JB):                     2.75e-07
Kurtosis:                       4.561   Cond. No.                         15.6
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
"""
```

In [16]:
```
sm.graphics.plot_fit(results,1)
plt.show()
```



(c) RMSE of GJR-GARCH forecasts

```
In [17]: RMSE = np.sqrt(((arr_realized_vol - arr_conditional_vol)**2).sum()/ \
                        len(arr_conditional_vol))

         RMSE

Out[17]: 2.8625505618637703
```

(d) find the best GARCH(p, q) model

```
In [18]: df_model_OS_results = pd.DataFrame(
             index=["BIC", "RMSE", "MAE", "MAPE"],
             columns=["GJR-GARCH({}, {})".format(p+1, q+1) for p in range(3)
                      for q in range(3)])

In [19]: for p in range(1, 4):
             for q in range(1, 4):
                 split_date = "2014-12-31"
                 am = arch_model(arr_monthly_ret[:split_date], p=p, o=1, q=q)
                 res = am.fit(disp="off")

                 garch_name = "GJR-GARCH({}, {})".format(p, q)
                 df_model_OS_results.loc["BIC", garch_name] = res.bic
```

```
        # print(res.summary())
        arr_ret_error = arr_monthly_ret[split_date:] - res.params["mu"]
        arr_realized_vol = np.abs(arr_ret_error)
        arr_conditional_vol = pd.Series(index=arr_realized_vol.index)
        # arr_conditional_vol[split_date] = arr_realized_vol[split_date]
        arr_conditional_vol[:max(p, q)] = arr_realized_vol[:max(p, q)]

        for i in range(max(p, q), len(arr_conditional_vol[split_date:])):
            step_sum = res.params["omega"]
            for j in range(p):
                step_sum += \
                    res.params["alpha[{}]".format(j+1)]*arr_realized_vol[i-1-j]**2
            for k in range(q):
                step_sum += \
                    res.params["beta[{}]".format(k+1)]*arr_conditional_vol[i-1-k]**2
            if arr_ret_error[i-1]<0:
                temp_sum += res.params["gamma[1]"]*arr_realized_vol[i-1]**2
            arr_conditional_vol[i] = np.sqrt(step_sum)

        df_model_OS_results.loc["RMSE", garch_name] = \
            np.sqrt(((arr_realized_vol - arr_conditional_vol)**2).sum()/ \
                len(arr_conditional_vol))
        df_model_OS_results.loc["MAE", garch_name] = \
            np.abs(arr_realized_vol - arr_conditional_vol).sum()/ \
                len(arr_conditional_vol)
        df_model_OS_results.loc["MAPE", garch_name] = np.abs(
            100*(arr_realized_vol - arr_conditional_vol)/arr_realized_vol).sum()/ \
                len(arr_conditional_vol)
```

```
In [20]: df_model_OS_results
```

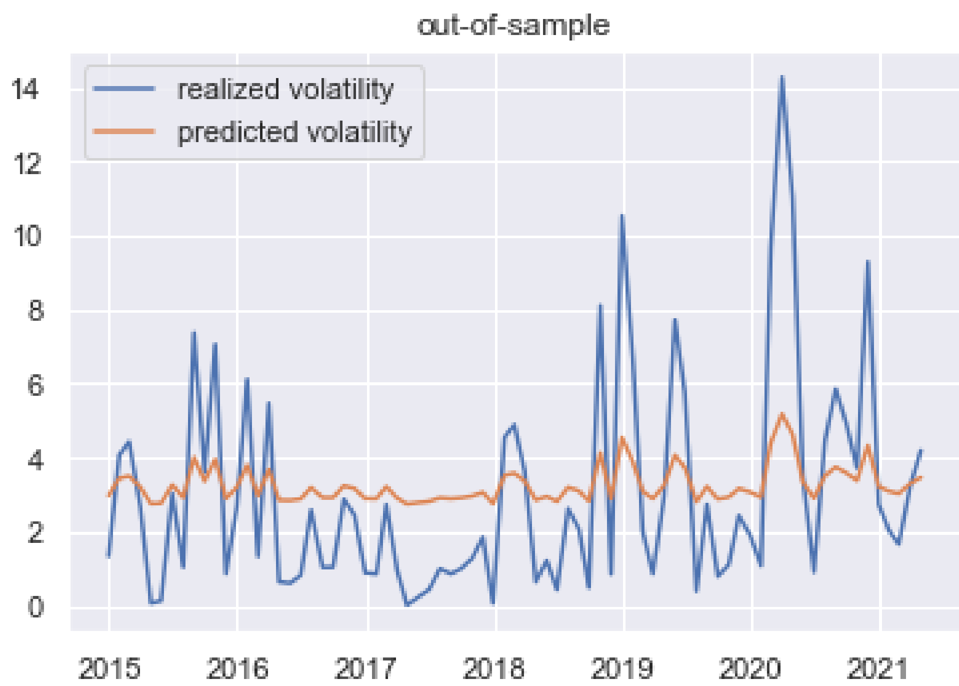
```
Out[20]:      GJR-GARCH(1, 1) GJR-GARCH(1, 2) GJR-GARCH(1, 3) GJR-GARCH(2, 1)  \
         BIC          3131.31         3132.05         3138.18         3132.05
         RMSE         2.80641         2.91919         2.91978         2.90454
         MAE          2.21098         2.31297         2.29032         2.28718
         MAPE         285.652         257.895         260.705         1219.51

              GJR-GARCH(2, 2) GJR-GARCH(2, 3) GJR-GARCH(3, 1) GJR-GARCH(3, 2)  \
         BIC          3136.66         3142.33         3133.25          3138.7
         RMSE         2.85928         2.85385         3.10443         2.94221
         MAE          2.23053          2.2074         2.51539         2.28927
         MAPE         659.002         352.983          290.32         357.748

              GJR-GARCH(3, 3)
         BIC          3131.34
         RMSE         3.05816
         MAE          2.39153
         MAPE          276.74
```

0.0.3 Q4

```
In [21]: # AR1 models (estimate constant vol)

         y = np.array(np.abs(arr_monthly_ret[:split_date] - arr_monthly_ret[split_date:].mean(
         reg = sm.OLS(y[1:], sm.add_constant(y[:-1])).fit()
         print(reg.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.029
Model:                            OLS   Adj. R-squared:                  0.027
Method:                 Least Squares   F-statistic:                     16.17
Date:                Wed, 05 May 2021   Prob (F-statistic):           6.61e-05
Time:                        21:54:55   Log-Likelihood:                -1344.5
No. Observations:                 539   AIC:                             2693.
Df Residuals:                     537   BIC:                             2702.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          2.7466      0.190     14.494      0.000       2.374       3.119
x1             0.1705      0.042      4.022      0.000       0.087       0.254
==============================================================================
Omnibus:                      264.327   Durbin-Watson:                   2.027
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1823.572
Skew:                           2.054   Prob(JB):                         0.00
Kurtosis:                      11.020   Cond. No.                         6.89
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
In [22]: arr_ret_error = arr_monthly_ret[split_date:] - arr_monthly_ret[split_date:].mean()
         arr_realized_vol = np.abs(arr_ret_error)
         arr_predicted_vol = pd.Series(reg.predict(sm.add_constant(np.array(arr_realized_vol))
                                       index=arr_realized_vol.index)

         plt.title("out-of-sample")
         plt.plot(arr_realized_vol, label="realized volatility")
         plt.plot(arr_predicted_vol, label="predicted volatility")

         plt.legend()

Out[22]: <matplotlib.legend.Legend at 0x1358307f0>
```



In [23]:
```
arr_realized_vol = np.array(arr_realized_vol)
arr_predicted_vol = np.array(arr_predicted_vol)

RMSE = np.sqrt(((arr_realized_vol[1:] - arr_predicted_vol[:-1])**2).sum()/ \
            len(arr_realized_vol))

MAE = (np.abs(arr_realized_vol[1:] - arr_predicted_vol[:-1])).sum()/ \
            len(arr_realized_vol)

print("RMSE is {}; MAE is {}".format(RMSE, MAE))
```

```
RMSE is 2.7897498871146706; MAE is 2.154831181506927
```

Out-of-sample comparison
RMSE: GARCH(1, 1) > GJR-GARCH(1,1) > AR(1)
MAE: GARCH(1, 1) > GJR-GARCH(1,1) > AR(1)

In [ ]:

Same process for the GE return, we can get similar conclusion,

In [24]:
```
# monthly log return
df_ge_ret["yearmonth"] = df_ge_ret.index.astype(str).str[:7]
df_ge_ret_monthly = df_ge_ret.drop_duplicates(subset=["yearmonth"], keep="last")
```

```
df_ge_ret_monthly["log_Close"] = np.log(df_ge_ret_monthly["Adj Close"])
df_ge_ret_monthly["log_return"] = 100*(df_ge_ret_monthly["log_Close"] - \
                                    df_ge_ret_monthly["log_Close"].shift(1))
```

In [25]:
```
arr_monthly_ret = df_ge_ret_monthly["log_return"].dropna()
```

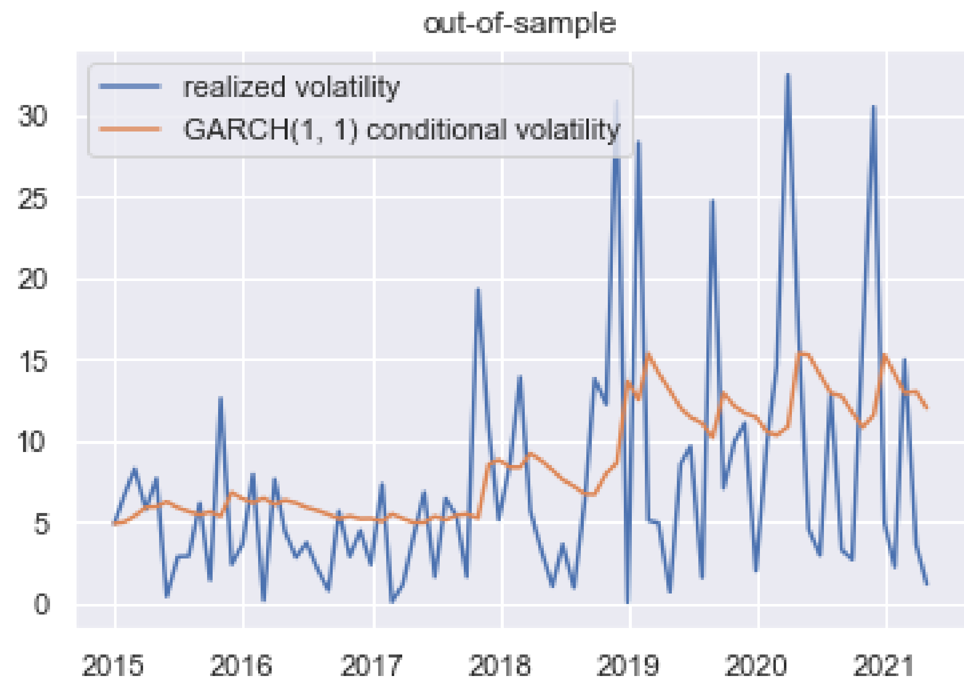
In [26]:
```
# out-of-sample forecast
split_date = "2014-12-31"
am = ConstantMean(arr_monthly_ret[:split_date])
# am = ConstantMean(arr_monthly_ret)
am.volatility = GARCH(1, 0, 1)
am.distribution = Normal()
res = am.fit(disp="off")

print(res.summary())
```

```
                     Constant Mean - GARCH Model Results
==============================================================================
Dep. Variable:             log_return   R-squared:                       0.000
Mean Model:             Constant Mean   Adj. R-squared:                  0.000
Vol Model:                      GARCH   Log-Likelihood:               -1775.41
Distribution:                  Normal   AIC:                           3558.81
Method:            Maximum Likelihood   BIC:                           3575.98
                                        No. Observations:                  540
Date:                Wed, May 05 2021   Df Residuals:                      539
Time:                        21:55:16   Df Model:                            1
                               Mean Model
========================================================================
                 coef    std err          t      P>|t|  95.0% Conf. Int.
------------------------------------------------------------------------
mu             1.0920      0.270      4.046  5.218e-05 [  0.563,  1.621]
                             Volatility Model
==========================================================================
                 coef    std err          t      P>|t|    95.0% Conf. Int.
--------------------------------------------------------------------------
omega          2.1702      0.844      2.572  1.010e-02   [  0.517,  3.824]
alpha[1]       0.1290  3.000e-02      4.300  1.710e-05 [7.019e-02,  0.188]
beta[1]        0.8277  2.946e-02     28.095 1.127e-173   [  0.770,  0.885]
==========================================================================

Covariance estimator: robust
```

In [27]:
```
arr_ret_error = arr_monthly_ret[split_date:] - res.params["mu"]
arr_realized_vol = np.abs(arr_ret_error)
arr_conditional_vol = pd.Series(index=arr_realized_vol.index)
arr_conditional_vol[split_date] = arr_realized_vol[split_date]

for i in range(1, len(arr_conditional_vol[split_date:])):
```

```
    arr_conditional_vol[i] = np.sqrt(res.params["omega"]
            + res.params["alpha[1]"]*arr_realized_vol[i-1]**2
            + res.params["beta[1]"]*arr_conditional_vol[i-1]**2)

# arr_conditional_vol = res.conditional_volatility
plt.title("out-of-sample")
plt.plot(arr_realized_vol, label="realized volatility")
plt.plot(arr_conditional_vol, label="GARCH(1, 1) conditional volatility")
plt.legend()
```

Out[27]: <matplotlib.legend.Legend at 0x135adfeb8>

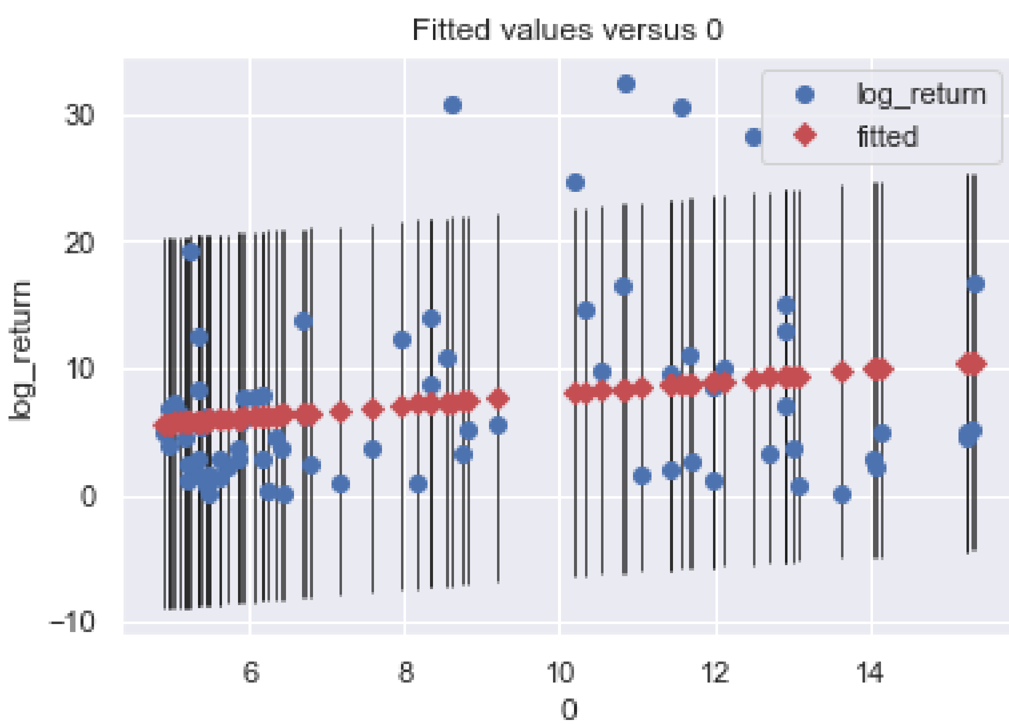


In [28]:
```
model = sm.OLS(arr_realized_vol, sm.add_constant(arr_conditional_vol))
results = model.fit(disp="off")
results.summary()
```

Out[28]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:             log_return   R-squared:                       0.046
Model:                            OLS   Adj. R-squared:                  0.033
Method:                 Least Squares   F-statistic:                     3.581
Date:                Wed, 05 May 2021   Prob (F-statistic):             0.0623
Time:                        21:55:20   Log-Likelihood:                -260.61
```

```
No. Observations:                  77   AIC:                             525.2
Df Residuals:                      75   BIC:                             529.9
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          3.3479      2.276      1.471      0.145      -1.186       7.881
0              0.4649      0.246      1.892      0.062      -0.024       0.954
==============================================================================
Omnibus:                       33.988   Durbin-Watson:                   2.029
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               63.106
Skew:                           1.685   Prob(JB):                     1.98e-14
Kurtosis:                       5.884   Cond. No.                         25.8
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
"""
```

In [29]:
```
sm.graphics.plot_fit(results,1)
plt.show()
```



In [30]:
```
RMSE = np.sqrt(((arr_realized_vol - arr_conditional_vol)**2).sum()/ \
            len(arr_conditional_vol))
```

```
RMSE
```

Out[30]: 7.470404489846029

```
In [31]: df_model_OS_results = pd.DataFrame(
             index=["BIC", "RMSE", "MAE", "MAPE"],
             columns=["GARCH({}, {})".format(p+1, q+1) for p in range(3)
                      for q in range(3)])

         for p in range(1, 4):
             for q in range(1, 4):
                 split_date = "2014-12-31"
                 am = ConstantMean(arr_monthly_ret[:split_date])
                 # am = ConstantMean(arr_monthly_ret)
                 am.volatility = GARCH(p, 0, q)
                 am.distribution = Normal()
                 res = am.fit(disp="off")

                 garch_name = "GARCH({}, {})".format(p, q)
                 df_model_OS_results.loc["BIC", garch_name] = res.bic

                 # print(res.summary())
                 arr_ret_error = arr_monthly_ret[split_date:] - res.params["mu"]
                 arr_realized_vol = np.abs(arr_ret_error)
                 arr_conditional_vol = pd.Series(index=arr_realized_vol.index)
                 # arr_conditional_vol[split_date] = arr_realized_vol[split_date]
                 arr_conditional_vol[:max(p, q)] = arr_realized_vol[:max(p, q)]

                 for i in range(max(p, q), len(arr_conditional_vol[split_date:])):
                     step_sum = res.params["omega"]
                     for j in range(p):
                         step_sum += \
                             res.params["alpha[{}]".format(j+1)]*arr_realized_vol[i-1-j]**2
                     for k in range(q):
                         step_sum += \
                             res.params["beta[{}]".format(k+1)]*arr_conditional_vol[i-1-k]**2
                     arr_conditional_vol[i] = np.sqrt(step_sum)

                 df_model_OS_results.loc["RMSE", garch_name] = \
                     np.sqrt(((arr_realized_vol - arr_conditional_vol)**2).sum()/ \
                         len(arr_conditional_vol))
                 df_model_OS_results.loc["MAE", garch_name] = \
                     np.abs(arr_realized_vol - arr_conditional_vol).sum()/ \
                         len(arr_conditional_vol)
                 df_model_OS_results.loc["MAPE", garch_name] = np.abs(
                     100*(arr_realized_vol - arr_conditional_vol)/arr_realized_vol).sum()/ \
                         len(arr_conditional_vol)
```

```
In [32]: df_model_OS_results
```

```
Out[32]:      GARCH(1, 1) GARCH(1, 2) GARCH(1, 3) GARCH(2, 1) GARCH(2, 2) GARCH(2, 3)  \
         BIC      3575.98     3578.94     3585.23     3581.67     3584.28     3590.57
         RMSE      7.4704     7.47695     7.47421     7.53465     7.47893     7.47641
         MAE       5.5226     5.31234     5.27759     5.58248     5.38772     5.35131
         MAPE     1019.13     917.668     923.507     3690.58     910.527      915.82

              GARCH(3, 1) GARCH(3, 2) GARCH(3, 3)
         BIC      3586.62     3587.62     3590.14
         RMSE     7.68542     7.62345     7.78692
         MAE      5.63636     5.51728     5.62788
         MAPE     837.012     895.705     5193.92
```

```
In [ ]:
```

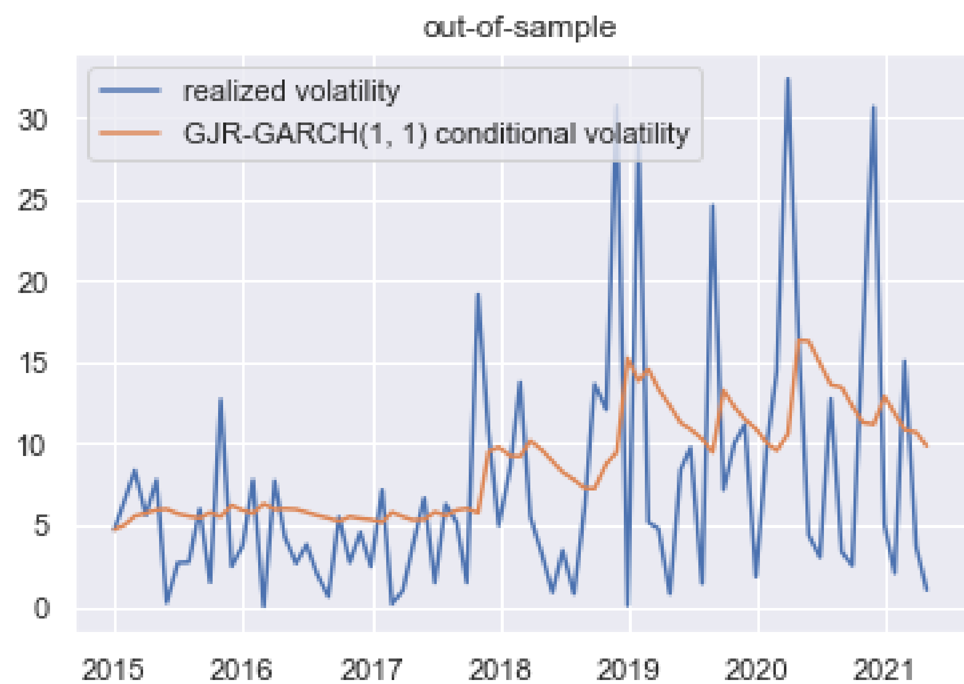
try GJR GARCH

```
In [33]: am = arch_model(arr_monthly_ret[:split_date], p=1, o=1, q=1)
         res = am.fit(disp="off")
         print(res.summary())
```

```
                   Constant Mean - GJR-GARCH Model Results
==============================================================================
Dep. Variable:             log_return   R-squared:                       0.000
Mean Model:             Constant Mean   Adj. R-squared:                  0.000
Vol Model:                  GJR-GARCH   Log-Likelihood:               -1773.39
Distribution:                  Normal   AIC:                           3556.78
Method:            Maximum Likelihood   BIC:                           3578.24
                                        No. Observations:                  540
Date:                Wed, May 05 2021   Df Residuals:                      539
Time:                        21:55:26   Df Model:                            1
                               Mean Model
=========================================================================
                 coef    std err          t      P>|t|   95.0% Conf. Int.
-------------------------------------------------------------------------
mu             0.9687      0.268      3.612  3.035e-04 [  0.443,  1.494]
                              Volatility Model
============================================================================
                 coef    std err          t      P>|t|      95.0% Conf. Int.
----------------------------------------------------------------------------
omega          3.2419      1.872      1.732  8.336e-02    [ -0.428,  6.911]
alpha[1]       0.0651  4.265e-02      1.526      0.127 [-1.850e-02,  0.149]
gamma[1]       0.0997  7.596e-02      1.313      0.189 [-4.915e-02,  0.249]
beta[1]        0.8147  4.789e-02     17.012  6.655e-65    [  0.721,  0.909]
============================================================================

Covariance estimator: robust
```

In [34]:
```
arr_ret_error = arr_monthly_ret[split_date:] - res.params["mu"]
arr_realized_vol = np.abs(arr_ret_error)
arr_conditional_vol = pd.Series(index=arr_realized_vol.index)
arr_conditional_vol[split_date] = arr_realized_vol[split_date]

for i in range(1, len(arr_conditional_vol[split_date:])):
    temp_sum = res.params["omega"] \
            + res.params["alpha[1]"]*arr_realized_vol[i-1]**2 \
            + res.params["beta[1]"]*arr_conditional_vol[i-1]**2
    if arr_ret_error[i-1]<0:
        temp_sum += res.params["gamma[1]"]*arr_realized_vol[i-1]**2

    arr_conditional_vol[i] = np.sqrt(temp_sum)

# arr_conditional_vol = res.conditional_volatility
plt.title("out-of-sample")
plt.plot(arr_realized_vol, label="realized volatility")
plt.plot(arr_conditional_vol, label="GJR-GARCH(1, 1) conditional volatility")
plt.legend()
```

Out[34]: <matplotlib.legend.Legend at 0x1356754e0>



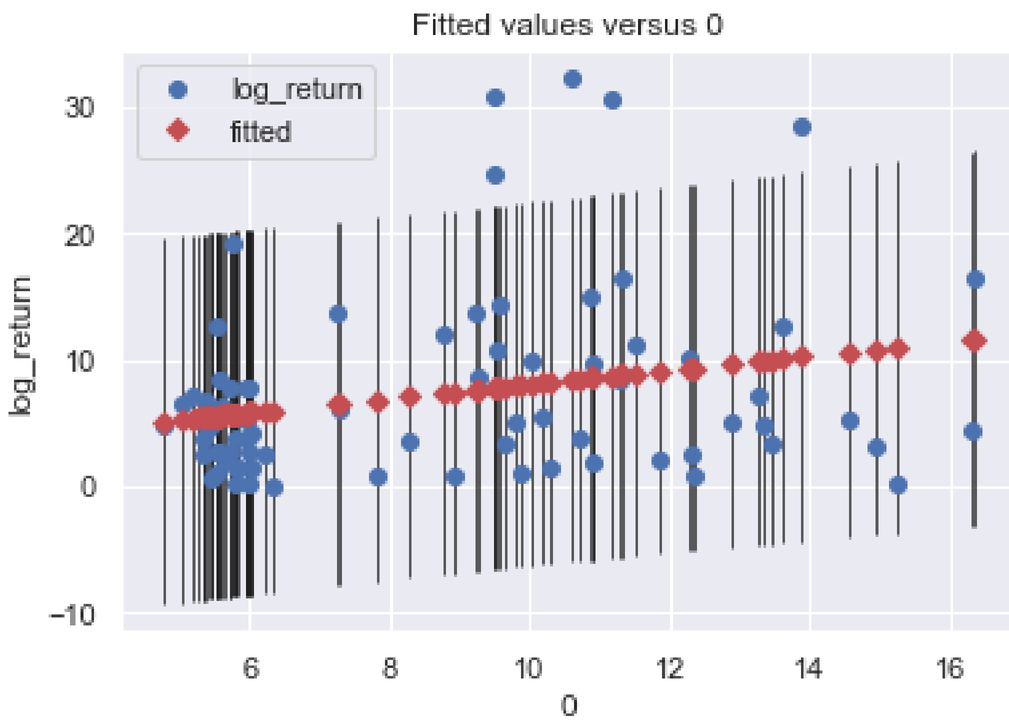
In [35]:
```
model = sm.OLS(arr_realized_vol, sm.add_constant(arr_conditional_vol))
results = model.fit()
results.summary()
```

Out[35]: <class 'statsmodels.iolib.summary.Summary'>

```
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:             log_return   R-squared:                       0.062
Model:                            OLS   Adj. R-squared:                  0.050
Method:                 Least Squares   F-statistic:                     4.970
Date:                Wed, 05 May 2021   Prob (F-statistic):             0.0288
Time:                        21:55:28   Log-Likelihood:                -259.99
No. Observations:                  77   AIC:                             524.0
Df Residuals:                      75   BIC:                             528.7
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          2.4500      2.336      1.049      0.298      -2.203       7.103
0              0.5615      0.252      2.229      0.029       0.060       1.063
==============================================================================
Omnibus:                       32.955   Durbin-Watson:                   2.082
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               60.057
Skew:                           1.640   Prob(JB):                     9.10e-14
Kurtosis:                       5.822   Cond. No.                         26.8
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
"""
```

In [36]:

```
sm.graphics.plot_fit(results,1)
plt.show()
```




```
In [37]: RMSE = np.sqrt(((arr_realized_vol - arr_conditional_vol)**2).sum()/ \
                    len(arr_conditional_vol))

         RMSE

Out[37]: 7.350501469111115

In [38]: df_model_OS_results = pd.DataFrame(
             index=["BIC", "RMSE", "MAE", "MAPE"],
             columns=["GJR-GARCH({}, {})".format(p+1, q+1) for p in range(3)
                      for q in range(3)])

In [39]: for p in range(1, 4):
             for q in range(1, 4):
                 split_date = "2014-12-31"
                 am = arch_model(arr_monthly_ret[:split_date], p=p, o=1, q=q)
                 res = am.fit(disp="off")

                 garch_name = "GJR-GARCH({}, {})".format(p, q)
                 df_model_OS_results.loc["BIC", garch_name] = res.bic

                 # print(res.summary())
                 arr_ret_error = arr_monthly_ret[split_date:] - res.params["mu"]
                 arr_realized_vol = np.abs(arr_ret_error)
```

```python
            arr_conditional_vol = pd.Series(index=arr_realized_vol.index)
            # arr_conditional_vol[split_date] = arr_realized_vol[split_date]
            arr_conditional_vol[:max(p, q)] = arr_realized_vol[:max(p, q)]

            for i in range(max(p, q), len(arr_conditional_vol[split_date:])):
                step_sum = res.params["omega"]
                for j in range(p):
                    step_sum += \
                        res.params["alpha[{}]".format(j+1)]*arr_realized_vol[i-1-j]**2
                for k in range(q):
                    step_sum += \
                        res.params["beta[{}]".format(k+1)]*arr_conditional_vol[i-1-k]**2
                if arr_ret_error[i-1]<0:
                    temp_sum += res.params["gamma[1]"]*arr_realized_vol[i-1]**2
                arr_conditional_vol[i] = np.sqrt(step_sum)

            df_model_OS_results.loc["RMSE", garch_name] = \
                np.sqrt(((arr_realized_vol - arr_conditional_vol)**2).sum()/ \
                    len(arr_conditional_vol))
            df_model_OS_results.loc["MAE", garch_name] = \
                np.abs(arr_realized_vol - arr_conditional_vol).sum()/ \
                    len(arr_conditional_vol)
            df_model_OS_results.loc["MAPE", garch_name] = np.abs(
                100*(arr_realized_vol - arr_conditional_vol)/arr_realized_vol).sum()/ \
                    len(arr_conditional_vol)
```

In [40]:
```python
df_model_OS_results
```

Out[40]:
```
     GJR-GARCH(1, 1) GJR-GARCH(1, 2) GJR-GARCH(1, 3) GJR-GARCH(2, 1)  \
BIC          3578.24         3582.84         3589.13         3581.81
RMSE         7.19211         7.25176         7.24878         7.27634
MAE          5.09499         5.08609         5.04614         5.10789
MAPE          530.88         533.241         542.955         384.955

     GJR-GARCH(2, 2) GJR-GARCH(2, 3) GJR-GARCH(3, 1) GJR-GARCH(3, 2)  \
BIC          3588.07         3594.02         3584.86         3585.57
RMSE         7.26603         7.27034         7.66385         7.46572
MAE          5.10192         5.10785         5.30934          5.1266
MAPE         393.536         453.414         459.726         355.085

     GJR-GARCH(3, 3)
BIC          3587.57
RMSE         7.66641
MAE          5.27949
MAPE         482.058
```

In [ ]:

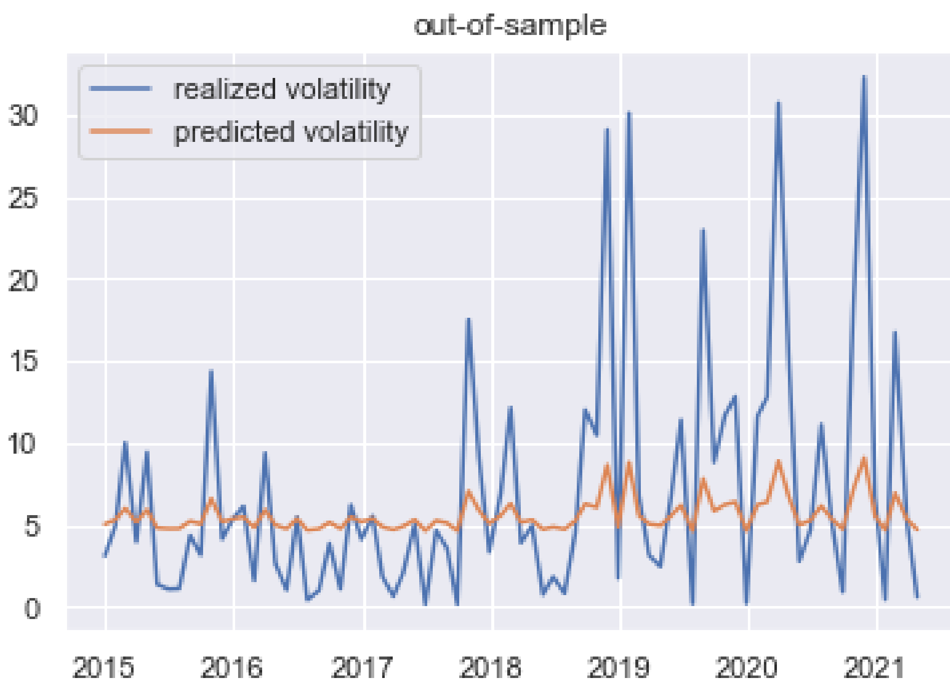
In [41]:
```python
# AR1 models (estimate constant vol)
y = np.array(np.abs(arr_monthly_ret[:split_date] - arr_monthly_ret[split_date:].mean()
```

```python
reg = sm.OLS(y[1:], sm.add_constant(y[:-1])).fit()
print(reg.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.019
Model:                            OLS   Adj. R-squared:                  0.017
Method:                 Least Squares   F-statistic:                     10.52
Date:                Wed, 05 May 2021   Prob (F-statistic):            0.00126
Time:                        21:55:37   Log-Likelihood:                -1597.8
No. Observations:                 539   AIC:                             3200.
Df Residuals:                     537   BIC:                             3208.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          4.6207      0.306     15.097      0.000       4.019       5.222
x1             0.1386      0.043      3.243      0.001       0.055       0.223
==============================================================================
Omnibus:                      158.829   Durbin-Watson:                   2.030
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              410.804
Skew:                           1.475   Prob(JB):                     6.24e-90
Kurtosis:                       6.096   Cond. No.                         10.9
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [42]:
```python
arr_ret_error = arr_monthly_ret[split_date:] - arr_monthly_ret[split_date:].mean()
arr_realized_vol = np.abs(arr_ret_error)
arr_predicted_vol = pd.Series(reg.predict(sm.add_constant(np.array(arr_realized_vol))
                              index=arr_realized_vol.index)

plt.title("out-of-sample")
plt.plot(arr_realized_vol, label="realized volatility")
plt.plot(arr_predicted_vol, label="predicted volatility")

plt.legend()
```

Out[42]:
```
<matplotlib.legend.Legend at 0x135f0b860>
```




```
In [43]: arr_realized_vol = np.array(arr_realized_vol)
         arr_predicted_vol = np.array(arr_predicted_vol)

         RMSE = np.sqrt(((arr_realized_vol[1:] - arr_predicted_vol[:-1])**2).sum()/ \
                     len(arr_realized_vol))

         MAE = (np.abs(arr_realized_vol[1:] - arr_predicted_vol[:-1])).sum()/ \
                     len(arr_realized_vol)

         print("RMSE is {}; MAE is {}".format(RMSE, MAE))
```

```
RMSE is 7.475000329474823; MAE is 4.967676469631952
```

RMSE: AR(1) > GARCH(1, 1) > GJR-GARCH(1,1)
MAE: GARCH(1, 1) > GJR-GARCH(1,1) > AR(1)

```
In [ ]:
```